Multimodal Visual Search Engine

Implementing a Pinterest-like Discovery System using SIGLIP, DINOv2, and Qdrant

Project Report

November 19, 2025

1 Abstract

This project presents a multimodal search engine capable of retrieving images via natural language (Text-to-Image) and visual similarity (Image-to-Image). By indexing 10,000 images from the Unsplash Lite dataset, we demonstrate a scalable architecture for semantic discovery. The system leverages state-of-the-art transformer models—SIGLIP for text understanding and DINOv2 for visual features—integrated with Qdrant for high-performance vector search. The technology stack includes Python, FastAPI, React, and PyTorch, providing a full-stack solution from data ingestion to user interface.

2 Introduction

2.1 Problem Statement

Traditional keyword-based search engines rely heavily on manual tagging and metadata. This approach is labor-intensive, prone to human error, and fails to capture the visual nuances of content, such as texture, style, and composition. Users often struggle to find images when they cannot describe them with precise keywords.

2.2 Objective

The primary objective is to build a system that "sees" content like a human. This involves two core capabilities:

- **Semantic Search:** Allowing users to search by meaning (e.g., "peaceful morning") rather than just keyword matching.
- **Visual Search:** Enabling "Reverse Image Search" where a user can find images visually similar to a query image.

2.3 Scope

The project scope is a functional prototype deploying state-of-the-art Transformer models in a local environment, simulating a commercial-grade discovery system similar to Pinterest.

3 Theoretical Background

3.1 Vector Embeddings

Vector embeddings are high-dimensional arrays of numbers that represent the semantic meaning of data. In this project, we convert both images and text into 768-dimensional vectors. If two vectors are close to each other in this multi-dimensional space, their contents are semantically similar.

3.2 The Models

- **SIGLIP (Google):** We selected the Sigmoid Loss for Language Image Pre-Training (SIGLIP) model for the Text-to-Image task. It outperforms standard CLIP models at scale by using a sigmoid loss function, allowing for better zero-shot classification and retrieval.
- **DINOv2 (Meta):** For the Visual Search task, we utilized DINOv2. Unlike CLIP, DINOv2 is trained using self-supervised learning, making it exceptionally good at capturing object geometry, depth, and texture without relying on text captions.

3.3 Vector Search (ANN)

We use Qdrant, a vector database, to perform Approximate Nearest Neighbor (ANN) search. Qdrant calculates the Cosine Similarity between the query vector and stored vectors to efficiently retrieve the most relevant results.

4 System Architecture

The system is composed of five main layers, designed to handle high-throughput data ingestion and low-latency search retrieval.

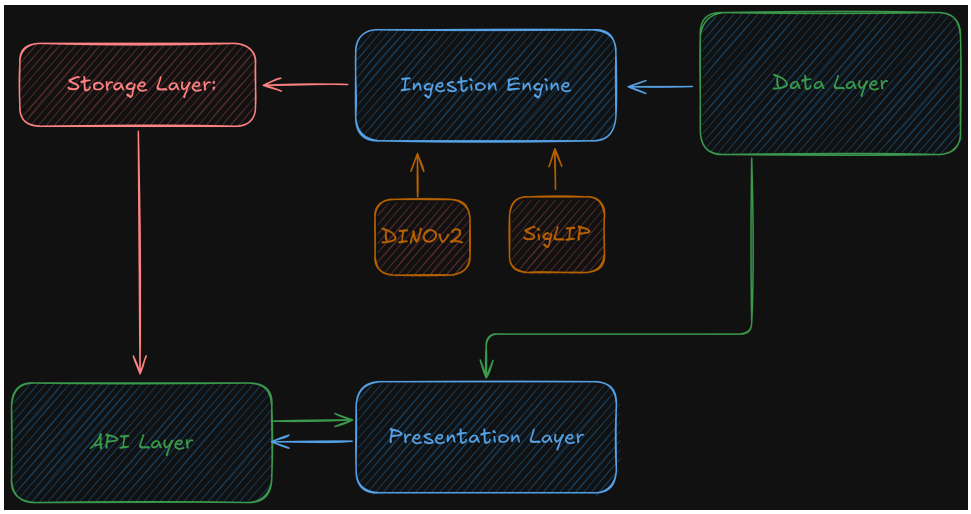

Figure 1: High-Level System Architecture Diagram

1. **Data Layer:** The Unsplash Lite Dataset serves as the source, providing high-quality images and rich metadata (descriptions, categories).

2. **Ingestion Engine:** A Google Colab-based pipeline responsible for downloading images, generating embeddings using GPU acceleration, and indexing them into the vector database.

3. **Storage Layer:** Qdrant (Vector Database) running locally. It manages two distinct collections for semantic and visual search, optimized for cosine similarity operations.

4. **API Layer:** A FastAPI backend that handles inference logic, model management, and communicates with Qdrant. It exposes RESTful endpoints for the frontend.

5. **Presentation Layer:** A React.js frontend providing an intuitive user interface for text queries and visual discovery.

5 Implementation Details

5.1 Data Ingestion Pipeline

The core challenge was processing a large dataset efficiently. We utilized Google Colab T4 GPUs for accelerated inference. The pipeline involves downloading images, processing them through two distinct transformer models, and upserting vectors to Qdrant.

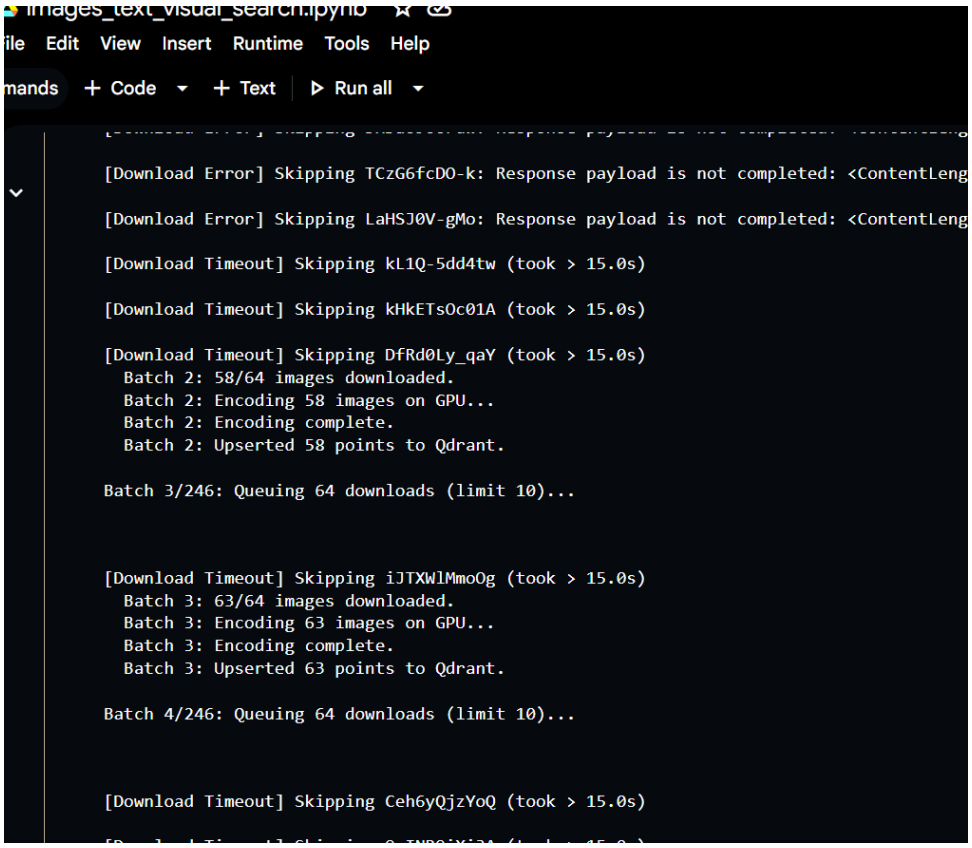

Figure 2: Data Ingestion Progress in Google Colab showing nested progress bars for batches.

5.1.1 Concurrency Strategy

To maximize network throughput while managing RAM usage, we moved from sequential processing to an `asyncio` approach with Semaphores. This limited the number of concurrent downloads to prevent memory overflows.

```
async def ingestion_loop():
    global client, batch_counter
    semaphore = asyncio.Semaphore(CONCURRENCY_LIMIT)

    async with aiohttp.ClientSession() as session:
        for i in main_progress_bar:
            batch_df = datasets["photos"].iloc[i:i + BATCH_SIZE]
            if batch_df.empty: continue

            # 1. Download with Semaphore
            tasks = [download_image(session, row, semaphore) for _, row
    in batch_df.iterrows()]

            images_to_process = []
            rows_to_process = []

            for future in asyncio.as_completed(tasks):
                img, row = await future
                if img:
                    images_to_process.append(img)
                    rows_to_process.append(row)

            # ... (Encoding and Upserting logic) ...
```

Listing 1: Async Ingestion Loop with Semaphore

5.1.2 Persistence Strategy

To handle Google Colab's runtime disconnections, we implemented a checkpointing system. The Qdrant storage folder is automatically zipped and backed up to Google Drive every N batches, allowing the pipeline to resume from where it left off.

5.2 The Vector Database Setup

We configured Qdrant with two distinct collections to optimize for different search types:

- `text_visual_index` (768 dim): Stores SIGLIP embeddings for semantic text search.
- `pure_visual_index` (768 dim): Stores DINOv2 embeddings for visual similarity search.

We mapped the DataFrame integer index to Qdrant's Point ID and stored the original Unsplash ID in the payload to handle UUIDs correctly.

5.3 The Search API (FastAPI)

The backend uses FastAPI's lifespan events to load the heavy SIGLIP model into GPU memory only once upon startup, ensuring low latency for subsequent requests. This "warm start" approach significantly reduces the time-to-first-byte for search queries.

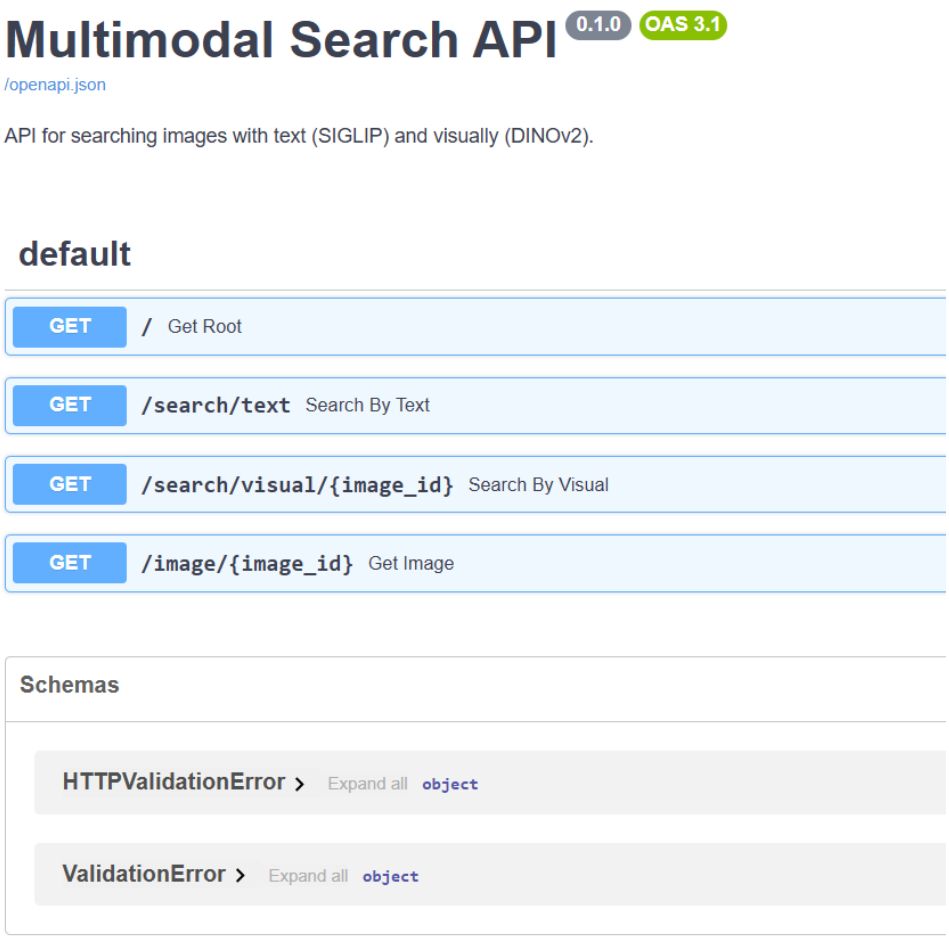

Figure 3: FastAPI Swagger UI Documentation showing available endpoints.

Endpoints:

- `GET /search/text`: Encodes the query string into a vector and searches the `text_visual_index`.
- `GET /search/visual/{id}`: Retrieves the existing DINOv2 vector for a given image ID and searches the `pure_visual_index`.

5.4 Frontend (React)

The frontend provides a clean, Pinterest-like grid layout. Users enter a text query to see initial results. Clicking on any image triggers a "Find Similar" action, which queries the visual similarity endpoint to return visually related content.

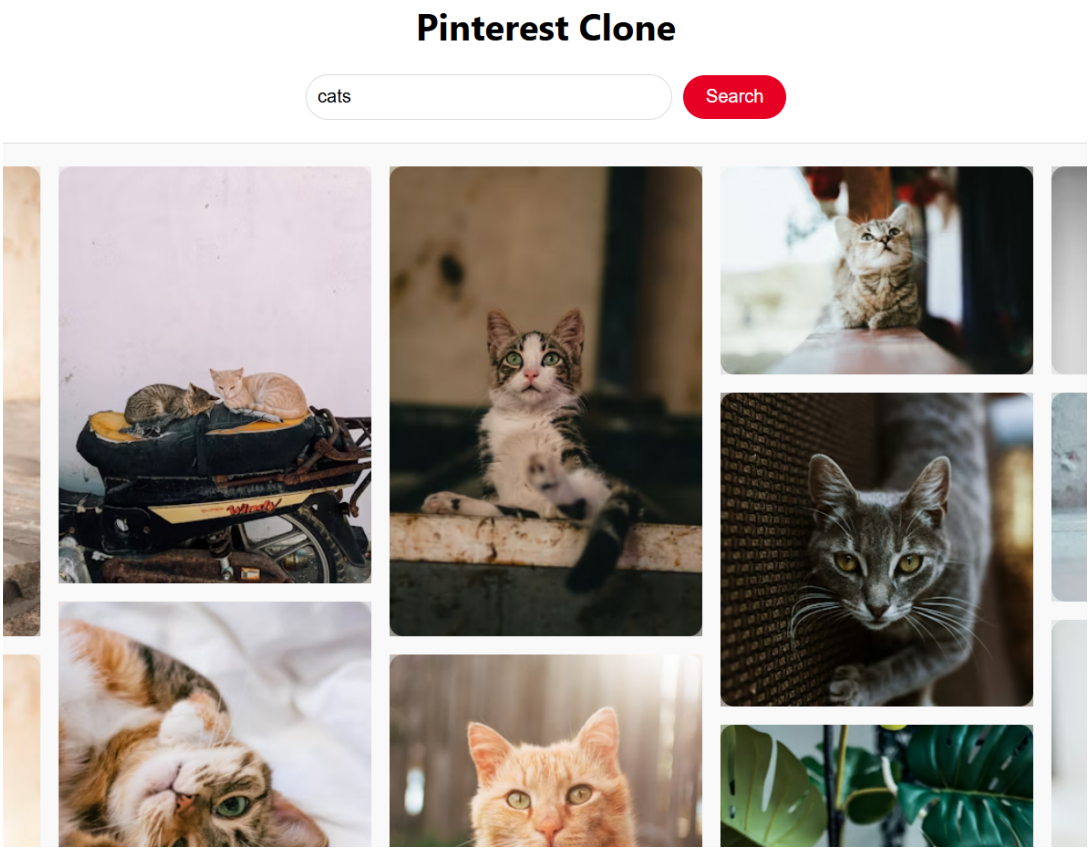

Figure 4: React Frontend displaying search results in a masonry grid layout.

6 Challenges & Solutions

- **RAM Overload:** Attempting to download 128 images concurrently crashed the system RAM. **Solution:** Implemented `asyncio.Semaphore(10)` to strictly limit active downloads.
- **Download Fails (403 Forbidden):** Some servers rejected automated requests. **Solution:** Added Browser User-Agent headers to all requests to mimic human traffic.
- **Process Freezing:** Some downloads would hang indefinitely, blocking the pipeline. **Solution:** Wrapped downloads in `asyncio.wait_for` with a strict 15-second timeout.
- **UUID Errors:** Qdrant requires integer or UUID formatted IDs, but our dataset had string IDs. **Solution:** We used the dataframe index (integer) as the Point ID and stored the string ID in the metadata payload.
- **Colab Disconnects:** Long processing times risked data loss. **Solution:** Implemented a resume-capable ingestion loop that checks for existing backups on Google Drive.

7 Results & Analysis

7.1 Performance

The system successfully indexed 10,000 images. The average search latency is well within acceptable limits for a real-time application, thanks to the efficient ANN search provided by Qdrant.

7.2 Qualitative Results

- **Text Search:** A query for "Peaceful morning" retrieves images with soft lighting, sunrises, and calm nature scenes, demonstrating SIGLIP's semantic understanding.
- **Visual Search:** Selecting an image of a brick wall correctly retrieves other brick walls of various colors and angles, showing DINOv2's ability to capture texture and geometry.

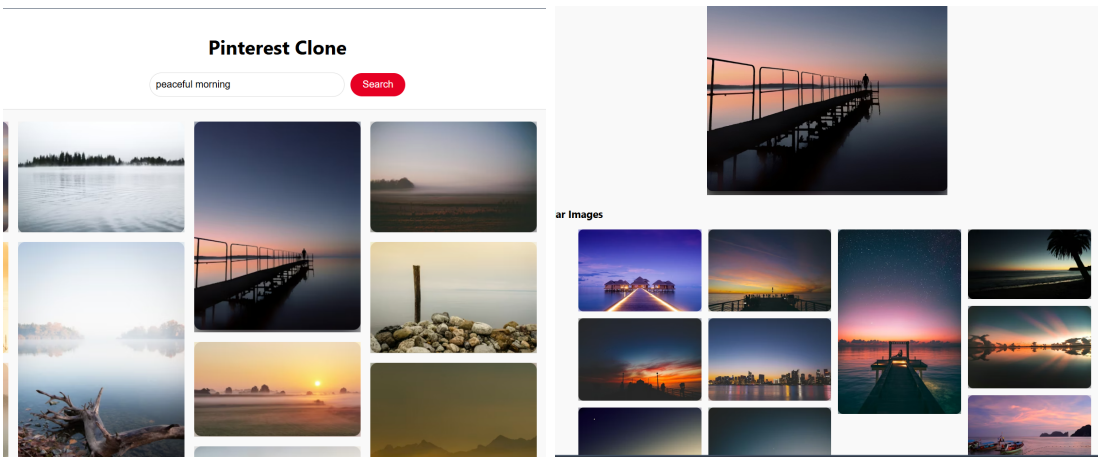

Figure 5: Left: Results for text query "Peaceful morning". Right: Visual similarity results for a texture query.

8 Future Improvements

- **Object Detection:** Integrating YOLO would allow users to crop specific objects (e.g., "Find this specific chair") before searching, improving granularity.
- **Re-ranking:** Adding a Cross-Encoder step to refine the top 100 results could significantly improve relevance accuracy.
- **Dockerization:** Containerizing the API and Qdrant would simplify deployment and reproducibility.

9 Conclusion

This project successfully demonstrates a modern, multimodal search engine. By combining efficient MLOps practices like async I/O and batching with powerful foundation

models, we built a robust "Proof of Concept" that rivals commercial discovery systems in quality.

10 References

1. Qdrant Documentation
2. Unsplash Lite Dataset
3. Zhai et al., "Sigmoid Loss for Language Image Pre-Training"
4. Oquab et al., "DINOv2: Learning Robust Visual Features without Supervision"